Rapid Authoring of Task Knowledge for Training and Performance Support

John L. Mohammed
Stottler Henke Associates, Inc.
San Mateo, CA
mohammed@stottlerhenke.com

Barbara Sorensen
Air Force Research Laboratory
Mesa, AZ
Barbara.Sorensen@mesa.afmc.af.mil

James Ong & Jian Li
Stottler Henke Associates, Inc.
San Mateo, CA
ong@stottlerhenke.com
li@stottlerhenke.com

ABSTRACT

Intelligent tutoring systems evaluate student performance and provide coaching and feedback during and/or after exercises. Intelligent job aids help users execute procedures by providing step-by-step instructions. These systems use computable task representations that specify appropriate actions at each step. These knowledge representations must be expressive enough to enable detailed, context-sensitive guidance and feedback, handle the wide range of situations and anomalies that might occur, and accurately assess the various possible actions the student might take. Yet, these representations must also enable easy and rapid knowledge entry and maintenance of large collections of procedures and training scenarios.

This paper describes an intelligent job aid and integrated simulation-based tutoring system developed for the Air Force to help satellite operators carry out complex command plans. These systems use hierarchical, object-oriented task representations that enable rapid authoring by non-programmers while supporting sophisticated job aiding and student performance evaluation. For example, the tutoring scenario editor enables the instructor to create an initial solution template by demonstrating a correct sequence of actions. The instructor can generalize this template, so the tutoring system can recognize alternate orderings of actions, alternative sets of actions that accomplish the same task, and conditional actions that are appropriate in certain situations.

The job aid helps users execute procedures by presenting step-by-step instructions using HTML-formatted text and graphics, hyperlinks, and embedded graphical user interface components. It enables *gradual automation* by presenting instructions to the operator for some steps while automating other steps by computing values, interpreting data, recommending actions, and sending and receiving information with other systems and databases. Looping and branching enable the software to execute some steps repeatedly or only when certain conditions are true. A graphical overview of the steps' hierarchical organization and flow-of-control helps operators and procedure authors quickly review and understand the procedure and maintain context during execution.

ABOUT THE AUTHORS

John Mohammed is a project manager at Stottler Henke. His research focuses on the application of artificial intelligence to space operations. His research for the US Air Force and NASA spans intelligent job aiding, simulation-based intelligent tutoring, model-based reasoning, automated anomaly resolution, fault diagnosis and recovery, and automated planning and scheduling of space-based systems. Dr. Mohammed led the design of an intelligent job aid and authoring tool designed to help US Air Force satellite operators execute complex command plans quickly and accurately. He also led the design of a software toolkit for rapidly developing scenario-based simulators and intelligent tutoring systems for satellite operations and other technical training areas. Before coming to Stottler Henke, Dr. Mohammed was a computer scientist at Schlumberger Palo Alto Research and the Fairchild Laboratory for Artificial Intelligence. Dr. Mohammed received a PhD in Computer Science from Stanford University. He has published 13 papers in refereed journals and conference proceedings.

Barbara Sorensen is a senior research scientist, US Air Force Research Laboratory Program manager and strategic advisor to USAF for the design and development of basic, exploratory and applied training research programs in advanced aircrew, command and control, and space training and simulation research. She designs and develops instructional and training technology across government, industry and academia for advanced biomedical, survivability and space-based capabilities and to support information and battle-space dominance, air superiority, mission rehearsal, distributed mission training, situational awareness, and modeling and simulation.

1. REPORT DATE **2005**	2. REPORT TYPE	3. DATES COVERED **-**	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Rapid Authoring of Task Knowledge for Training and Performance Support	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Research Laboratory,Mesa,AZ,85212**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**12**	

James Ong is a researcher and group manager at Stottler Henke. His work focuses on intelligent tutoring systems in areas such as undersea acoustic analysis, NASA payload operations, and satellite operations. He also leads the development of software that enables rapid review and exploration of multivariate, time-oriented data using high-density, interactive, graphical displays. James has held engineering, engineering management, applied research, and marketing positions at Stottler Henke, AT&T Bell Laboratories, Bolt Beranek and Newman, and Belmont Research. James received an MS degree in electrical engineering and computer science from U.C. Berkeley, an MS degree in computer science (artificial intelligence) from Yale University, and an MBA from Boston University.

Jian Li is a software engineer at Stottler Henke. He led the implementation of an intelligent job aid and authoring tool that provides step-by-step guidance and partial automation to support satellite operations and other procedural and semi-procedural tasks. He also led the implementation of a scenario-based intelligent tutoring system and authoring tool for technical training, as well as high-density graphical data display components for reviewing multivariate, time-oriented data. He has also contributed to the development of training and education systems that teach helicopter piloting and math problem-solving.

Rapid Authoring of Task Knowledge for Training and Performance Support

John L. Mohammed
Stottler Henke Associates, Inc.
San Mateo, CA
mohammed@stottlerhenke.com

Barbara Sorensen
Air Force Research Laboratory
Mesa, AZ
Barbara.Sorensen@mesa.afmc.af.mil

James Ong & Jian Li
Stottler Henke Associates, Inc.
San Mateo, CA
ong@stottlerhenke.com
li@stottlerhenke.com

INTRODUCTION

Increasingly, military and commercial satellite systems are employing constellations of satellites in low earth orbit (LEO) for communications and remote sensing. Satellite system management is complicated by the large number of satellites to be managed and the brief time windows when each satellite is visible to ground communication sites during which communication can take place. Therefore, it is essential that operators make the best use of every opportunity to communicate with each satellite as it comes into view. Electronic job aids can help operators execute complex procedures more quickly and reliably by generating and presenting step-by-step instructions and by automating steps when appropriate. In addition, extensive simulation-based training with instructional feedback can prepare students with repeated practice and exposure to a wide range of nominal and off-nominal situations.

This paper describes an electronic job aid and a simulation-based intelligent tutoring system developed by the authors to support satellite operations and other complex procedural tasks. These systems rely on computable task representations that specify appropriate actions at each step. These task representations must be expressive enough to enable detailed, context-sensitive guidance and feedback, handle the wide range of situations and anomalies that might occur, and accurately assess the many possible actions the student might take. Yet, they must also enable easy and rapid knowledge entry and maintenance of large collections of procedures and training scenarios.

SCENARIO-BASED INTELLIGENT TUTORING

The advantages of scenario-based training are well known (Schank, 1995). The student practices performing tasks in a realistic simulation of the operational environment, receives exposure to a variety of nominal and unusual situations, and gets an opportunity to see how classroom knowledge is applied in context. Simulated scenarios are also a critical part of evaluating student performance for certification.

Intelligent tutoring systems (ITSs) can significantly improve the effectiveness of scenario-based training by providing instructional feedback that helps students learn from their experiences more reliably. ITSs can track the student's progress during the execution of a training scenario. They can be configured to give *in situ* coaching during exercises such as hints and detailed instructions for what to do, how to do it, and why. ITSs can also assess the student's actions, identify areas of strong and weak performance and provide feedback after the student completes the scenario. ITSs enable each student to receive individualized training that would normally require the full attention of a human tutor -- without requiring one instructor per student. ITSs also enable the student's training to proceed at a pace that is suitable for that particular student. By reducing the need for specialized equipment and team members during training exercises, it can also provide increased flexibility regarding when and where training takes place.

Intelligent tutoring systems (ITSs) encode and apply the subject matter teaching expertise of experienced instructors to provide students with individualized instruction automatically. For procedural skills such as executing satellite command plans, this expertise includes task knowledge that enables the ITS to evaluate the appropriateness of the students' actions and assess their knowledge and skills.

To support training for satellite operations and other procedural tasks, we enhanced a tutoring system and authoring tool called the Task Tutor Toolkit that was originally developed for NASA to support remote payload operations and other technical training areas (Ong and Noneman, 2000). This system encodes task knowledge as *scenario-specific solution templates* that encode allowable sequences of actions for each scenario.

During each exercise, the simulator uses the tutoring system's application programming interface (API) to notify the tutoring system of each student action. The simulator also provides query access to simulation state variable values that the tutor can consider when determining the appropriateness of each student action. Each action is encoded as a tuple that specifies the type of action and zero or more parameters. For example, setting the oven temperature to 300 degrees might be represented as:

```
(set-control "temperature" 300)
```

In this example, set-control is the type of action. Two parameters, "temperature" and 300 specify the type of control and the setting, respectively.

Hinting

At each step, the student can request hints by pressing buttons in the tutoring system window:

- **Give me a hint** – The tutoring system provides an indirect hint that helps the student determine an appropriate next action to take.
- **What do I do?** – The tutoring system recommends an appropriate action.
- **How do I do that?** – The tutoring system describes how the student should carry out the recommended action using the simulator.
- **Why do I do that?** – The tutoring system explains why the recommended action should be taken. This explanation may be scenario-specific, or it may describe general principles associated with the recommended action.

Evaluating Student Actions

The tutoring system evaluates each action by comparing it with the scenario's solution template. After each action taken by the student, the system displays whether the student's action was:·

- **Expected** - the action matches an action pattern in the solution template, and the student has already carried out all prerequisite actions that should precede this action. For example, an action pattern might match the setting of the temperature control to any value between 290 and 310 degrees.
- **Unexpected** - the action does not match any action pattern in the solution template, or the action has already been carried out, or not all prerequisite steps have been carried out. When a student carries out an unexpected action, that action may change the state of the simulated world in a way that invalidates the template's

expectations for appropriate next steps. In these situations, the solution template may become invalid, and the tutoring system may no longer be able to assess subsequent student actions.

- **Continuable –** the action is unexpected but benign, so the action did not change the state of the simulated world in a way that invalidated the solution template's expectations. The student can proceed with the scenario, and the tutor can continue to rely on the solution template to correctly evaluate subsequent actions.
- **Incorrect** - the action and current simulation state match an action pattern and simulation condition, if any, specified within an error rule.

Instructional Strategies for Procedural Training

By classifying each student action into one of these categories, the tutoring system can support several different instructional strategies. For example, a tutor could accept only expected and continuable actions and reject unexpected and incorrect actions by notifying the student and then instructing the simulator to *undo* the last action. Or, a tutor could accept all types of actions. Because the solution template's expectations might have been invalidated by an inappropriate action, however, the tutor would not be able to assess the subsequent actions reliably. However, as long as the simulation is able to behave realistically in response to subsequent actions, this instructional approach still gives students an opportunity to realize their mistake and experience their effects. For example, experiencing the simulated loss of a satellite due to operator error can be a motivating and memorable learning experience. Afterwards, the tutor could ask questions that prompt the student to reflect on his or her actions to figure out when the error was made, what the correct action should have been, and what the impact of the error was on the satellite or ground systems.
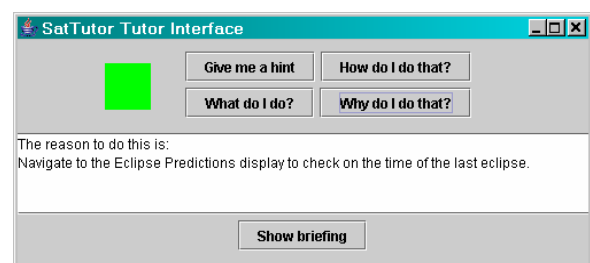

Figure 1 - The tutoring system enables the student to ask for context-sensitive hints during exercises

Task Representations for Tutoring

A key design issue for any tutoring system is the manner in which task knowledge is represented, or encoded, in a computable format that can be interpreted by the software. The task representation must be expressive enough to enable the tutor to assess each action and distinguish appropriate actions from inappropriate ones, even when there is more than one correct set of actions for a given scenario. The representation must also enable the tutoring system to assess the student's knowledge and skills and provide useful coaching and feedback during and after each exercise. Finally, the representation must enable rapid and intuitive knowledge entry by subject matter experts so that tutoring scenarios can be created easily and economically, without complex programming.

We chose to encode each solution template as a hierarchy of simple task nodes and group task nodes that represent the set of possible sequences of student actions that are appropriate for a scenario. Each **simple task node** recognizes a correct student action. It specifies:

- an *action pattern* that specifies the action type and constraints on its parameters. An action is expected (and appropriate) if its type matches that action pattern's type and its parameters satisfy the action pattern's constraints.

- an optional *simulation state condition* that specifies constraints on the values of simulation state variables that must be satisfied in order for the task node's action to be active and enabled for matching against incoming student actions.

- optional *principles* (typically, specific skills or pieces of knowledge) that are demonstrated when the student carries out an action that matches the action pattern when the node's simulation state condition is satisfied, and

- optional text strings that are displayed when the student requests the various types of *hints* associated with each step.

Each group task node contains:

- one or more simple task nodes and/or lower level group task nodes, and

- zero or more principles that are demonstrated when the student carries out all of the actions that are recognized by the simple task nodes and sub-group task nodes in the group.

Demonstrating, Generalizing, and Annotating Tutoring Scenarios

Instructors and subject matter experts (scenario authors) use the simulator to first **demonstrate** one (of possibly many) correct sequence of actions for the scenario. The *tutoring scenario editor* records these actions to create an initial solution template that recognizes this exact set of actions performed in order.

Scenario authors then use the tutoring scenario editor to **generalize** this solution template so that it recognizes other valid sequences of actions. For example, the author can relax constraints on the action's parameters by specifying multiple valid values or ranges of numeric values. The author can relax ordering constraints by specifying that the actions in a group of actions can be carried out in any order.· Or, the author can specify alternate sub-sequences of actions within a solution template. This feature enables the tutoring system to determine when the student carries out one of the several possible ways of performing a task within a scenario. Authors can also specify conditional actions that are appropriate only when certain simulation state conditions are true, expressed as a Boolean expression that refers to simulation state variables and, optionally, the action's parameters.

Authors then **annotate** the solution template by associating principles with actions or groups of actions. This enables the tutoring system to assign credit to the student for principles he or she appears to know when the action or group of actions is carried out.

INTELLIGENT JOB AIDS

Currently, document-based procedures or command plans present step-by-step instructions that guide satellite and ground station operators through the execution of satellite contacts. The main advantage of this approach is that the documents can be produced by non-programmers using familiar word processing software. A limitation of this approach is that the documents can only present instructions to the operator, but they cannot actually help the operator execute those instructions. The operator is still responsible for operating the mission operations software, by navigating its screens, requesting and interpreting information, performing calculations, constructing and issuing commands; and determining the appropriate next step in the document to execute.

Electronic job aids have the potential for reducing operator errors and increasing execution speed.

Some satellite operations systems use scripts to execute procedures or commands automatically. This approach works best when complete automation is feasible and algorithms exist that can assess the situation and make correct decisions in all situations. When this is not true, some operator control (or at least active participation) is necessary so that the operator can apply his or her knowledge and judgment to the situation. In these situations, the software and the operator share responsibility for carrying out the procedure, so it is necessary for the job aid software to present and prompt for information using effective user interfaces. In addition, the job aid must provide a scripting capability that complements rather than replaces the operator's judgment and skills.

An Intelligent Job aid for Procedural Tasks

We developed an intelligent job aid and authoring tool called TaskGuide to enable the Air Force to create and edit computable *procedure specifications* that help users carry out complex procedural tasks quickly and accurately. The job aid is comprised of a *Procedure Execution Tool* that is used by operators to run procedures and a *Procedure Editor* that is used by procedure authors to create and edit procedures.

The Procedure Execution Tool's user interface shown below contains three window panes. The Procedure Summary Pane in the upper left area provides a graphical summary of the steps and their hierarchical

organization to help operators and authors quickly browse and understand the procedure and keep track of where they are in the procedure during execution.

The Node Details Pane at right shows instructions for the step that is currently selected in the Procedure Summary Pane (during browsing) or the step that is currently being executed. It presents each step's instructions using HTML-formatted text, graphics, input controls, hyperlinks, and interactive graphical user interface components. Input controls such as text fields, check boxes, radio buttons, and selection lists prompt the operator for data, decisions, and requests. The job aid stores user input values in variables, so they can be referenced in calculations and test conditions in downstream steps and groups. Hyperlinks make additional information easily available on demand to augment each step's instructions. Instructions can also embed arbitrary graphical user interface components, implemented using the Java programming language and software libraries. This capability makes it possible to incorporate sophisticated, application-specific interactive displays.

After completing each step, the user presses the green arrow button to advance to the next step. The Procedure Execution Tool then determines and displays the appropriate next step according to the procedure's branching and looping logic. The Execution Log Pane in the lower left area lists each step that has been executed.
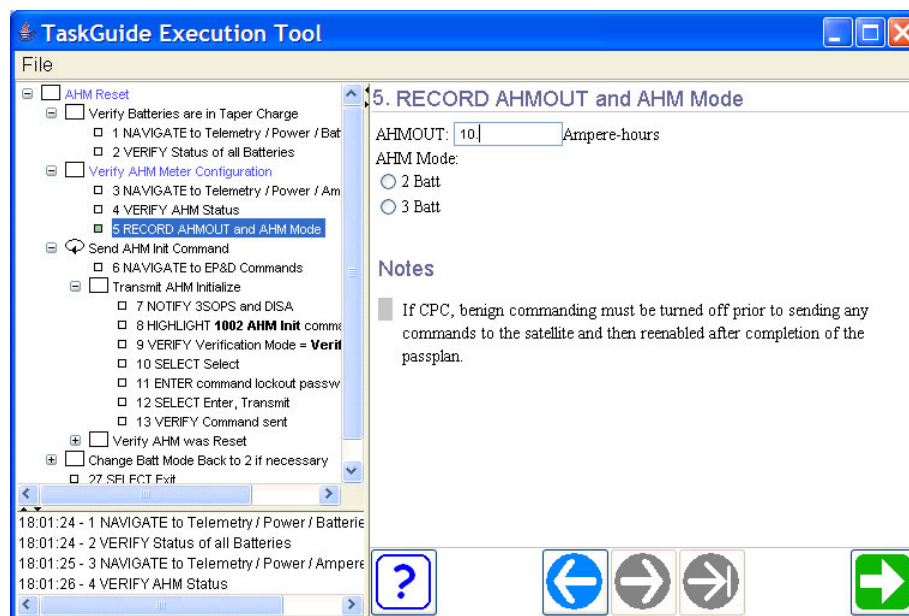

Figure 2 - The Procedure Execution Tool summarizes the procedure in the upper left pane and shows details of the selected step in the right pane.

The Procedure Summary Pane displays an icon and label for each step and group of steps. Different icons represent different types of groups and steps as shown in the tables below.

	Interactive	Automated
Simple Step	□	■
Exit Step	◇	◆

Simple Group	☐
Branching Group	↱
Loop Group	↻

Table 1 – Icons for each type of step node and group node in the procedure summary pane

Each simple step presents instructions or other types of information to the user and optionally prompts the user for input. An exit step has an exit condition that determines how the job aid advances to the next step. If the condition is true, the job aid exits from the group that contains the exit step. Otherwise, it advances to the next step in the usual way. Branching groups contain steps that are executed only if a test condition is true, and loop groups are executed repeatedly while a test condition is true.

The Procedure Summary Pane exploits the hierarchical organization of the task representation to present a graphical summary of the procedure that supports browsing, so operators can rapidly become familiar with (or refresh their memory of) the procedure. This pane uses indentation to show that a step or group lies within a higher-level group, similar to the way the Windows Explorer file browser displays files and folders. If a group icon is *collapsed*, the group's children are hidden. To expand a group and show its children, operators double-click on the group node's icon.

Operators can select a step or group by clicking on its icon. The details of the selected step or group are then displayed in the Step Details Pane. By reviewing higher-level groups before expanding them to see the details of lower-level groups and individual steps, operators can quickly browse large, complex procedures and understand the procedure's overall organization and logic before delving into its details.

The Procedure Summary Pane also helps the operator maintain context. During execution of a procedure, the TaskGuide Procedure Summary Pane highlights the *current step* being executed by displaying its icon and the background of its short description green.

Levels of Automation

The level of automation that is appropriate for a particular operation depends on several factors. First, automation of an operation requires that a reliable algorithm has been designed that correctly retrieves and interprets relevant information, makes decisions based on that information, and executes correct decisions in all situations. Automation is not feasible if the job aiding system cannot access some of the relevant data. For example, some of the relevant information might reside in the heads of other personnel, accesssible only via verbal communications. Or, some data that is ordinarily accessed by an operator using the user interface of a satellite operations system might not be available to a software system via inter-systems communication, due to a lack of systems integration. For some operations, even if an algorithm can perform well in nominal cases, human judgment and experience may be required to perform the operation correctly in exceptional cases, so reliable automation might not be possible in all situations.

For these reasons, it may be desirable to automate some operations in a procedure and rely on manual execution or manual review/override for others. In addition, over time, it may be possible to automate more and more of the operations within a procedure as reliable automation algorithms are developed and become trustworthy. Thus, is it highly desirable that any electronic job aid system for satellite operations be able to support varying levels of automation in a procedure and enable automation to be introduced gradually into a procedure to provide complete control over the degree of automation employed.

Our job aid supports three levels of automation. In *manual execution mode*, the job aid reduces operator workload by determining the appropriate step to carry out and by presenting instructions for the current step to the operator. Dynamically-generated instructions can further reduce the operator's cognitive load by presenting succinct instructions that are specific to the current situation, rather than static instructions that are necessarily more verbose so they can cover all possible situations.

The second mode is *manual review and override*. In this mode, the job aid automatically determines the next actions to be performed and describes this action to the operator so that the operator can accept or modify the action before it is executed.

The third mode is *automatic execution*. In this mode the job aid automatically performs the action required by the step without interaction with the operator.

Automated actions can include simple calculations based on data recorded by the operator or retrieved automatically from other components of the mission operations software, automated decision support (such as resource re-planning to contend with contingencies), and automated invocation of operations supported by the mission operations software. A single procedure can use all levels of automation. Some operations within the procedure may require manual operation, while others may use manual review/override or automated execution.

Task Representations for Job Aiding

When designing the system's task representations, we decided that they should resemble the step-by-step instructions as they are commonly presented in the Air Force's document-based command plans. This resemblance enables the job aid to support largely manual operations carried out by the operator by presenting step-by-step instructions like document-based instructions, when desired. Second, the task representation should enable the specification of queries, calculations, and commands to automate operations as deemed appropriate by the Air Force for each command plan. We achieved this goal by enabling *calculations*, or script-like program statements, to be run at the beginning and at the end of each step. A third goal was that the job aid should be able to communicate each step's instructions and provide additional information on demand in the most effective manner. Finally, the task representation should employ features of modern programming languages, such as hierarchical grouping of steps, conditional branching logic, and looping logic, to help procedure authors specify procedures that are understandable, error-free, and easily browsed using the Procedure Editor and Procedure Execution Tool.
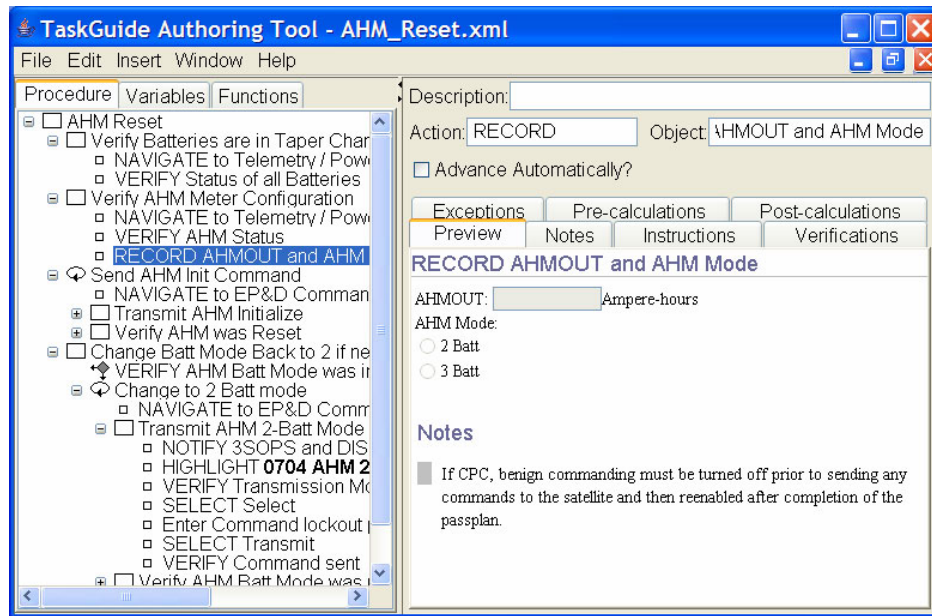

Figure 3 – The Job Aid Procedure Editor presents an overview of the procedure in the left pane and enables editing of the selected step or group in the right pane

Procedure Editor

A procedure specification encodes step-by-step instructions and execution logic as a list of steps, organized within a hierarchy. Each step contains HTML-formatted instructions that tell the operator what to do or prompt the operator for input. Steps can also present optional *verifications* that tell the user how to confirm successful completion of the step, as well as *notes* that describe conditions that must be maintained or avoided during the step, cautions and warnings, and other types of additional information. Authors can associate each note with a single step or with a group of steps. When a note associated with a group is displayed for the first time, a colored icon next to the note indicates that the note is new. When the note is displayed within later steps in the group, a gray icon indicates that the note has been displayed within previous steps.

The Procedure Editor shown above enables procedure authors to create procedure specifications that are executed by the Procedure Execution Tool. The left pane contains tabbed windows that display the procedure's steps and groups, along with the variables and functions that can be used within the

procedure. The right pane enables authors to edit the step or group that has been selected in the left pane.

Each step's instructions and verifications can either be *static* (canned) or it can be *generated dynamically*. A procedure can contain a mix of static and dynamically-generated instructions. In general, however, most instructions in a procedure specification are static and present the same information each time the procedure is executed. Procedure authors specify the content and format of static instructions as text and HTML tags. The Procedure Editor provides wizards that help authors create lists, tables, text fields, input controls, and other types of HTML tags.

Authors can specify dynamically-generated instructions by embedding expressions within the instruction's HTML text. During execution, the Procedure Execution Tool generates the instruction dynamically by evaluating each embedded expression and replacing it with its value. Expressions can contain references to variables whose values are entered by the user, received from external systems and databases during procedure execution, or derived from other variables using calculations. Compared to static instructions, dynamically-generated instructions can filter information to present instructions that are more succinct and targeted to the specific situation. They can also generate recommendations and compute default values for input parameters based on data already gathered.

Steps can also contain *calculations* that evaluate expressions containing constants, variables, and function calls and save these values in variables. These variable values can be used within calculations in downstream steps to send/receive data to/from other systems and databases, analyze and interpret this data, recommend actions to be taken by the user, or select and execute actions automatically. *Pre-calculations* execute at the beginning of each step (pre-calculations), before instructions are presented to the user. This is useful for retrieving and computing data or text strings so they can be embedded within dynamically-generated instructions. *Post-calculations* execute at the end of the step, after the user has followed the step's instructions, entered data, and indicated completion. This is useful for interpreting, processing, saving, or acting upon the user's inputs. Post-calculations can also contain error-checking statements that verify the user's input. If the input fails error-checking, the job aid reprompts the user for input by displaying the step's instructions and input controls again. Calculations can invoke standard math, Boolean, and string operations as well as arbitrary Java methods, enabling complex decision

support and interoperability with general purpose and application-specific software libraries.

Each step can be either *interactive* or *automated*. If the step is interactive, the job aid performs the step's pre-calculations (if any), presents the next step's instructions to the user, waits for the user to indicate completion of the step, and then performs the step's post-calculations (if any). If the step is automated, the job aid performs the step's calculations without displaying instructions or interacting with the user.

The job aid supports *gradual procedure automation*, so manual steps within procedures can be replaced, over time, with steps that retrieve data, compute values, and carry out actions automatically. The procedure author can specify the desired level of user awareness and override capability for each step. For example, an interactive step could use calculations to compute a default parameter value or decision and prompt the user to confirm or override it. As confidence in the reliability and robustness of the calculation increases, the organization could replace the interactive step with an automated step that uses a computed value or decision to perform an action without user intervention. In this manner, a manual procedure can evolve into a more automated one.

The job aids *extensible architecture* enables integration with both general purpose and application-specific software libraries that provide functions that are invoked by calculations. This architecture enables procedure specifications to incorporate arbitrarily complex automated data retrieval, interpretation, automated reasoning and decision-making algorithms. For example, optional systems integration with the satellite missions operations system would enable the procedure's calculations to receive data from the mission operations system and help the operator interpret this data, make decisions, construct satellite commands, and send these commands to the mission operations system for uploading and execution.
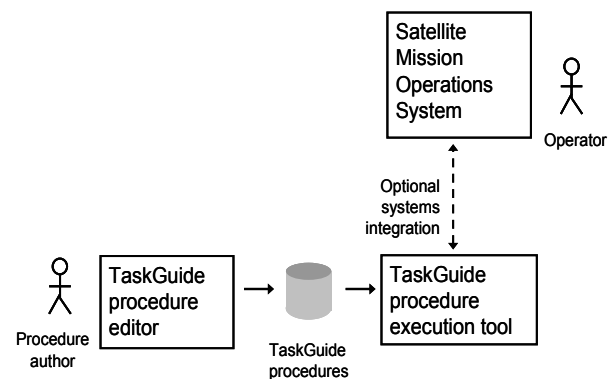

Figure 4 – Job aid data flow

INTEGRATED JOB AIDING AND SIMULATION-BASED TUTORING

We developed an integrated training system that combines a satellite operations simulator, the job aid, and the tutoring system. As shown in Figure 5, graphical editors enable entry and editing of tutoring scenarios and procedure specifications.

We developed a software framework for rapidly developing partial, scenario-specific simulations of the mission operations software, the ground station hardware and software and the satellite. The simulations are partial in that they only implement the parts of the simulated software's graphical user interface (GUI) that are relevant to each scenario. Screenshots of the actual mission operations software provide a realistic look, and interactive controls are overlaid on the screenshots only for those GUI

controls that will be acted upon by the student during the scenario or that must display scenario-specific data that changes over the course of the scenario.
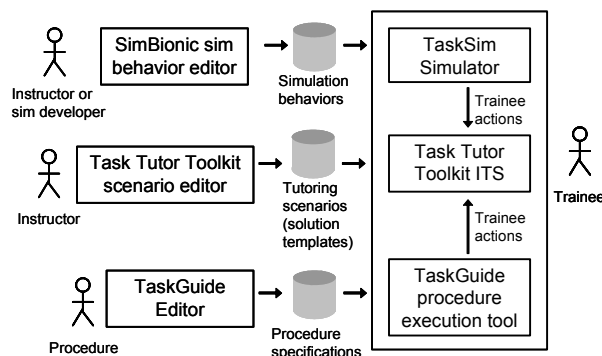

Figure 5 - Integrated simulation-based tutor and job aid data flow
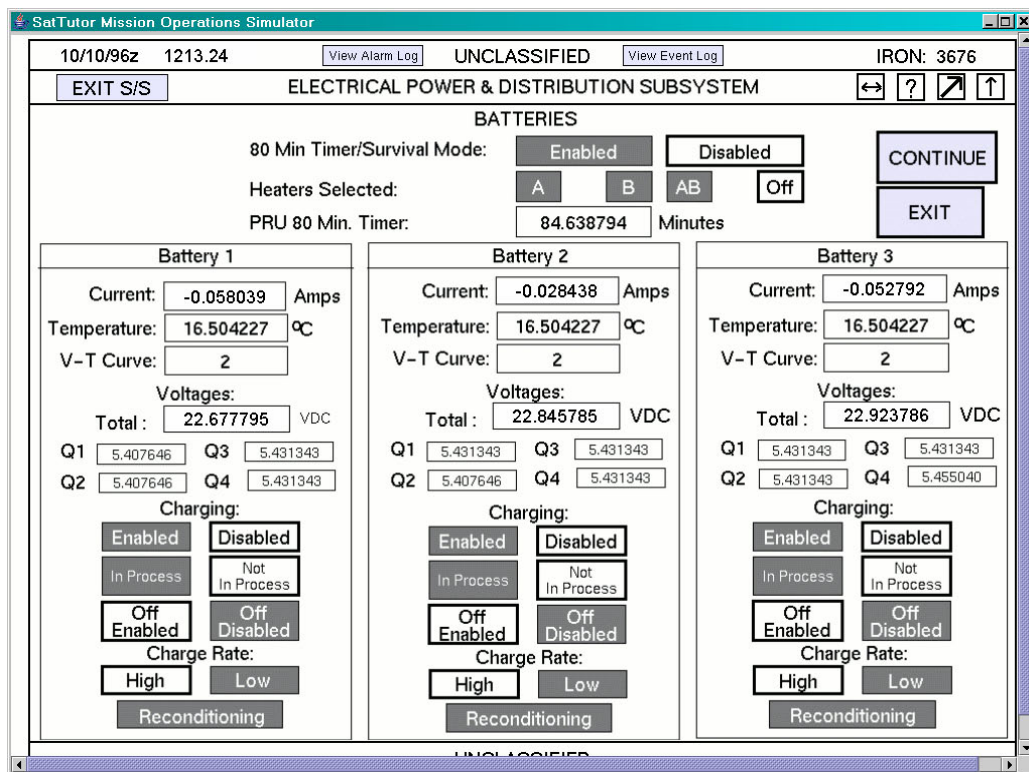

Figure 6 - Rapid development of scenario-specific simulations is enabled by using screen captures of the satellite operations system's user interface and selective implementation and overlay of the user interface controls that are likely to be used during the scenario

In general, it is costly and difficult to specify how a simulation of a complex system behaves in response to arbitrary student actions and other events. Our system avoids this problem by employing scenario-specific simulation behavior models that are valid only within a narrower envelope of the situations that

are likely to occur during a given scenario, rather than any possible action or event. This approach makes it possible to quickly create scenario-specific simulations that respond realistically to those actions the student is likely to perform. This actions include correct actions as well as incorrect actions that are

common or can be anticipated. A graphical editor enables scenario authors to quickly specify simulation behaviors as flow chart-like hierarchical behavior transition networks.

PRELIMINARY EVALUATION AND FUTURE WORK

The system was presented and demonstrated to 10 satellite operations instructors at Vandenberg AFB in February 2005. The reaction of the participants to the software was generally positive. During the presentation and demonstration, the instructors identified enhancements to the software that they felt were the most important for acceptance of the software for operations and training.

Six participants filled out an evaluation questionnaire comprised of 22 questions that prompted each respondent to rate the usefulness or usability of various aspects of the knowledge editors and run-time systems for the job aid, tutoring system, and training simulation. The average rating for all questions and respondents was 3.9 on a scale of 1 (hard to use, not effective or intuitive) to 5 (easy to use, very effective or intuitive). 98% of the ratings were between 3 and 5. Average ratings across the three systems were comparable, ranging from 3.65 for the simulation development tool, 3.88 for the job aid, and 4.08 for the tutoring system.

The questionnaires also prompted the respondents for open-ended comments regarding the most-useful/beneficial features, the most needed enhancements, and barriers to operational use. Most comments were positive regarding the software's capabilities and ease-of-use.

The respondents also identified additional job aid capabilities that might be needed to support operational use, such as:

- **Flexible execution**: the software should enable the operator to adapt procedure execution to accommodate anomalies and correct errors in real time, even in ways that are not anticipated in the procedure specification. For example, this might include backing up to redo parts of a procedure, skipping parts.
- **Rapid recovery from software/hardware failure:** the software should be able to quickly resume execution of a procedure interrupted by failure of the hardware/software running the software in order to ensure highly available monitoring and control of the satellite.

- **Incremental persistent store:** the software should incrementally save a record of each step's execution in a persistent store, such as a database, to support recovery and review of the procedure's execution log.

We have also identified other promising candidate enhancements to the job aid, such as integration with the site-specific workflow methods and software infrastructure; the ability to help operators keep track of elapsed time, time windows, and deadlines during procedure execution; and support for multi-person procedures.

We also identified potentially useful enhancements to the tutoring system. Currently, if the student carries out an unexpected action that is not benign, the solution template can no longer be assumed to accurately represent the next possible actions that the student should carry out. This is because non-benign unexpected actions may have altered the state of the world in a way that renders the solution template invalid. However, in many cases, it is possible to *recover* from an unexpected action by carrying out one or more additional actions that restore the state of the world so that procedure execution can proceed. To support recovery from unexpected actions that are not benign, it would be desirable to enhance the tutoring system to support *recoverable* actions. When the student performs a recognized recoverable action, the tutoring system can inform the student and guide him or her through a set of steps that recover from this action. This feature would enhance the realism and naturalness of the simulation-based exercise.

RELATED WORK

Studies show that individualized instruction provided by intelligent tutoring systems are highly effective. However, a barrier to their widespread use is the cost and difficulty of encoding the subject matter and instructional expertise used by the tutoring software, especially when "deep" representations of the task are used, such as full-blown planning-style representations (e.g. Sacerdoti, 1977; Rickel, et al., 2000), and cognitive models in production-system formats (e.g. Anderson, et al., 1990) that enable the tutor to act as an expert system in the task area. Scenario-specific task representations avoid the complexity and expertise needed to build an expert system (Murray, 1998). Authoring specific scenarios allows for focus on situations and decision points that are judged to be particularly important, and for highly tuned student assessment and instructional interventions. For example, Guralnik (1996) describes an authoring tool that applies a content

theory of procedural task knowledge, enabling the tutoring system to generate replies to important questions from the student. The work described in this paper builds upon prior work in software tutors for procedural training by us (Ong and Noneman, 2000) and others (Guralnik, 1996). Specifically, we enhanced the expressiveness of the task representations used by the tutor with constructs such as conditional actions, alternate actions, and continuable actions while striving to keep the task representations simple enough to be authored by non-programmers using graphical tutoring scenario editors.

There have been a number of systems developed to assist or automate the execution of procedural tasks. Most of these systems automate task execution by providing specialized scripting languages. For example, Timeliner (Busa, 2002) was developed by Draper Laboratories as a tool to automate procedural tasks on the International Space Station. These tasks may be sequential tasks that would typically be performed by a human operator, or precisely ordered sequencing tasks that allow autonomous execution of a control process. However, Schwarz et al claim that a combination of automation, fully-manual control, and human supervisory control generally yields the optimum level of automation in terms of system reliability and life cycle costs, including up-front development and operations costs.

Our approach differs in its focus on supporting partially-automated procedure specifications that combine sophisticated information presentation and user interface capabilities for interactive operations with scripting for automated operations. This approach provides greater flexibility and control over each procedure's use of automation and the division of labor between the operator and the software.

OTHER APPLICATIONS

This tutoring system and job aid can also be used to provide training and performance support for other technical tasks in which the number of appropriate ways of carrying out each task is limited. For example, these systems can help maintenance technicians diagnose and repair equipment, and they can help people operate equipment, use software applications, or perform tasks in compliance with organizational guidelines and procedures.

ACKNOWLEDGEMENTS

REFERENCES

Anderson, J.R., Boyle, C.F., Corbett, A. T., & Lewis, M.L. (1990). Cognitive Modeling and Intelligent Tutoring. *Artificial Intelligence* 42(1): 7--49.

Busa, J., E. Braunstein, R. Brunet, R. Grace, T. Vu and R. Brown. (2002) *Timeliner: Automating Procedures on the ISS*. SpaceOps, Houston, TX, October 9-12, 2002. Sponsored by AIAA.

Guralnik, D. (1996) An Authoring Tool for Procedural-Task Training". PhD Dissertation - Technical Report #71. *The Institute for the Learning Sciences, Northwestern University.*

Murray, T. (1998). Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design*, Journal of the Learning Sciences*, 7(1).

Ong, J and S. Ramachandran. (Feb 2000). Intelligent Tutoring Systems: The What and the How. *Learning Circuits on-line magazine.* Web: http://www.learningcircuits.org/2000/feb2000/ong.html.

Ong, J., S. Noneman (November 2000), Intelligent Tutoring Systems for Procedural Task Training of Remote Payload Operations at NASA, *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC 2000)*.

Rickel, J., Ganeshan, R., Rich, C., Sidner, C.L., & Lesh, N. (2000). Task-Oriented Tutorial Dialog: Issues and Agents. Mitsubishi Electric Research Laboratories Technical Report TR-2000-37.

Sacerdoti, E.D. (1977). A Structure for Plans and Behavior. American Elsevier, New York.

Saito, T., Ortiz, C., Mithal, S., & Loftin, R.B. (1991). Acquisition, Representation and Rule Generation for Procedural Knowledge, *Proceedings of the Second CLIPS Conference*, NASA/JSC, Houston, TX.

Schank, R. (1995). What We Learn When We Learn by Doing. *Technical Report no. 60*, Institute of Learning Sciences, Illinois.

Swartout, M., Kitts, C., & Batra, R., "Persistence-Based Production Rules for On-Board Satellite Automation".

Schwarz, R., Kuchar, C., Hastings, D., Deyst, J., Kolitz, S., A Probabilistic Model for the Determination of the Effects of Automation of Satellite Operations on Life Cycle Costs. Web: http://www.mit.edu/~jkkuchar/munich/munich.html